--24. A process for assigning tasks in a multiprocessor digital data processing system having a preemptive operating system, and a given number of processors capable of processing said tasks in parallel, comprising dividing said processors (20*a*-21*a*, 20*b*-22*b*, 20*c*) in at least one preliminary phase into groups (G*a*, G*b*, G*c*) each group comprising predetermined numbers of processors, and each of said processor groups being associated with an elementary queue (5*a*, 5*b*, 5*c*), each of the tasks being associated with one of the processors associated with said elementary queue (5*a*, 5*b*, 5*c*), and storing a predetermined number of tasks to be processed in a given order of priority.

A2

25. A process according to claim 24, characterized in that said groups each comprise an identical number of processors (200-203, 210-213).

Sub B1

26. A process according to claim 24, comprising generating a series of tests and measurements in an additional preliminary phase for determining the number of processors in each group and the number of groups for achieving the best performance of said system.

27. A process according to claim 24, wherein the architecture of said system is of the non-uniform memory access type (NUMA), and the system (1) is constituted by a predetermined number of modules (*M*0, *M*1) linked to one another, each comprising a given number of processors (200-203, 210-213) and storage means, each of said modules (*M*0, *M*1) constituting one of said groups, each module being associated with one of said elementary queues of an associated processor.

28. A process according to claim 24, further comprising associating each of said processors with a first data structure for identification of the associated processor, said first data structure comprises at least one first set of pointers (*p*200 through *p*203), associating said first set of pointers with one of said elementary queues (5*a*, 5*b*), associating each of said elementary queues (5*a*, 5*b*) with a second data structure, said second data structure having at least one second set of pointers (*pp*5*a*, *pp*5*b*), associating said second data structure with one of said

processor groups (200-201, 202-203), storing all of the tasks to be processed (*T*1 through *T*10) in said system (1) in a table (4), each of said second data structures of the elementary queues (5*a*, 5*b*) further comprising a third set of pointers (*pT*1, *pT*5, *pT*10), said third set of pointers each associating elementary queues (5*a*, 5*b*) with one of said tasks (*T*1 through *T*10) stored in the table (4) or with a series of concatenated tasks, and associating each of said tasks (*T*1 through *T*10) of the table (4) with a third data structure that comprises a fourth set of pointers (*p*5*a*1 through *p*5*a*4, *p*5*b*1 through *p*5*b*10) said fourth set of pointers associating third data structure with one of said elementary queues (5*a*, 5*b*).

A2

Sub B1

29. A process according to claim 24, further comprising distributing said tasks among said elementary queues (5*a*, 5*b*) in at least one additional phase by searching, when a new task to be processed (Tz) is created, for a queue with the lightest load (5*y*) among all of said elementary queues (5*a*, 5*x*, 5*y*, 5*p*) of said system (1) and assigning said new task to said elementary queue with the lightest load so as to balance the global load of said system (1) among said elementary queues (5*a*, 5*x*, 5*y*, 5*p*).

30. A process according to claim 29, further comprising performing said distribution of tasks by determining a composite load parameter associated with each of said elementary queues (5*a*, 5*x*, 5*y*, 5*p*) associating each processor (2*a*, 2*x*, 2*y*, 2*p*) with a memory (Mem*a*, Mem*x*, Mem*y*, Mem*p*), calculating said composite load parameter as the sum of the load of a processor or a processor group associated with said elementary queue and the load of the memory associated with said processor or processor group.

31. A process according to claim 29, further comprising checking in a preliminary step whether said task (Tz) is linked to one of said elementary queues (5*a*, 5*x*, 5*y*, 5*p*), and when said test is positive, assigning said linked task to the elementary queue.

32. A process according to claim 24, further comprising at least one additional phase and searching for a remote elementary queue (5*y*) that is not empty when one of said elementary

queues (*5q)* associated with one of said processor groups *(2q)* is empty of executable tasks selecting in said empty elementary queue (5*y*) a task executable by one of said processors (2*q*) of said processor group associated with the empty elementary queue (5*q*) and transmitting said selected task to said one of said processor (2*q*) for processing so as to globally balance the processing of said tasks in said system (1).

33. A process according to claim 32, characterized in that said non-empty elementary queue (5*y*) has a predetermined minimal occupation threshold.

A2 Sub B1

34. A process according to claim 33, further comprising storing the tasks in decreasing order of priority, skipping a predetermined number of tasks before scanning the other tasks of said non-empty elementary queue (5*y*) in order to search for an executable task and have said executable task processed by one of said processors (2*q*) of said processor group associated with the empty elementary queue (5*q*).

35. A process according to claim 34, characterized in that said number of skipped tasks and the maximum number of scanned tasks among all tasks stored in said non-empty elementary queue (5*q*) are variable over time and are determined by a self-adapting process from the number of tasks that are or are not found during said scans and from the position of these tasks, sequenced in order of priority, in said non-empty elementary queue (5*q*).

36. A process according to claim 32, characterized in that said selected task is associated with a minimal value of a cost parameter, which measures global performance degradation of said system (1) due to the processing of said selected task in said non-empty remote elementary queue (5*q*) by one of said processors of said processor group associated with the empty elementary queue (2*q*).

37. A process according to claim 24, further comprising periodically measuring for a balanced distribution of said tasks in said elementary queues (5*a*, 5*x*, 5*y*, 5*p*) in at least one

additional phase and when an unbalanced state of said system (1) is determined, selectively moving tasks from at least one elementary queue with a heavier load (5*x*) to an elementary queue with a lighter load (5*y*).

38. A process according to claim 37 comprising discontinuing the step of selectively moving tasks when said imbalance is below a certain threshold.

39. A process according to claim 37 wherein all or some of said tasks belong to multitask processes, and each multitask process requires a given memory size and workload, further comprising measuring workloads and memory sizes, in the system and selecting the process requiring the greatest workload and the smallest memory size, and moving all the tasks of said selected process to the elementary queue with the lightest load (5*y*).

Sub B1

40. A process according to claim 39, characterized in that it comprises a preliminary step of checking whether all tasks of said multitask process that must be moved belong to the elementary queue set with the heaviest load (5*x*) and whether any task is linked to any of said groups.

41.. A process according to claim 24 characterized in that said preemptive operating system is of the "UNIX" type.

42. Architecture for a multiprocessor digital data processing system comprising a given number of processors for implementing a process for assigning tasks to be processed to said processors, said system having a preemptive operating system and a given number of processors capable of processing said task in parallel, said processors (20*a*-21*a*, 20*b*-22*b*, 20*c*) being divided into groups (G*a*, G*b*, G*c*),and an elementary queue (5*a*, 5*b*, 5*c*) associated with each of the groups (G*a*, G*b*, G*c*), each of said elementary queues (5*a*, 5*b*, 5*c*) storing a predetermined number of tasks to be processed in a given order of priority, so that each of the

tasks of each of said elementary queues (5*a*, 5*b*, 5*c*) is associated with one of the processors of this elementary queue (20*a*-21*a*, 20*b*-22*b*, 20*c*).

43. Architecture according to claim 42, further comprising means (6) for determining the load of said elementary queues (5*a*, 5*x*, 5*y*, 5*p*) and for assigning a new task created in said system to the elementary queue with the lightest load (5*y*).

Sub B1

44. Architecture according to claim 42, further comprising, when one (5*q*) of said elementary queues (5*a*, 5*x*, 5*y*, 5*p*) associated with one of said processors (2*q*) is empty, means (7) for locating a non-empty, remote elementary queue (5*y*), and an executable task in said non empty elementary queue (5*y*), and assigning said executable task to said one of said processor (2*q*) for processing said executable task.

A2

45. Architecture according to claim 42, further comprising means (8) for detecting an imbalance between elementary queues (5*a*, 5*x*, 5*y*, 5*p*), and for determining when an imbalance is detected the elementary queue with the heaviest load (5*x*) and the elementary queue with the lightest load (5*y*), and means for moving tasks from the elementary queue with the heaviest load (5*x*) to the elementary queue with the lightest load (5*y*).

46. Architecture according to claim 42, wherein the operating system of the processing system is of the nonuniform memory access type (NUMA), and comprises modules (*M*0, *M*1) linked to one another, each module comprising a given number of processors (200-203, 210-213) and storage means, each of said modules (*M*0, *M*1) constituting one of said groups, each module (*M*0, *M*1) being associated with one of said elementary queues.

47. Architecture according to claim 43, wherein the operating system of the processing system is of the nonuniform memory access type (NUMA), and comprises modules (*M*0, *M*1) linked to one another, each module comprising a given number of processors (200-

203, 210-213) and storage means, each of said modules (*M*0, *M*1) constituting one of said groups, each module (*M*0, *M*1) being associated with one of said elementary queues.

Sub B1

48. Architecture according to claim 44, wherein the operating system of the processing system is of the nonuniform memory access type (NUMA), and comprises modules (*M*0, *M*1) linked to one another, each module comprising a given number of processors (200-203, 210-213) and storage means, each of said modules (*M*0, *M*1) constituting one of said groups, each module (*M*0, *M*1) being associated with one of said elementary queues.

A2 concl.

49. Architecture according to claim 45, wherein the operating system of the processing system is of the nonuniform memory access type (NUMA), and comprises modules (*M*0, *M*1) linked to one another, each module comprising a given number of processors (200-203, 210-213) and storage means, each of said modules (*M*0, *M*1) constituting one of said groups, each module (*M*0, *M*1) being associated with one of said elementary queues.--